REMARKS

Claims 1, 5, 7, 10, 16, 18, 23, 25, 27, 38, 43-44, 50-51, and 57-65 have been amended. Claims 1-65 remain pending in the application. Reconsideration is respectfully requested in light of the following remarks.

Section 101 Rejection:

The Office Action rejected claims 57-63 under 35 U.S.C. § 101, asserting that the claims are directed to allegedly non-statutory subject matter. As indicated above, these claims have been amended to recite a non-transitory, computer-readable storage medium, as suggested by the Examiner. Therefore, Applicants request removal of the rejection of claims 57-53 under 35 U.S.C. § 101.

Section 103(a) Rejections:

The Examiner rejected claims 1, 2, 4, 5, 13-16, 18, 19, 21-23, 30, 33-44, 49-51 and 56-65 under 35 U.S.C. § 103(a) as being unpatentable over Huppenthal, et al. (U.S. Patent 6,339,819) (hereinafter "Huppenthal") in view of Hinds, et al. (U.S. Patent 6,542,916) (hereinafter "Hinds"), and further in view of Chen et al. (U.S. Patent 6,763,365) (hereinafter "Chen"), claims 3, 20, 45 and 52 as being unpatentable over Huppenthal, Hinds and Chen, and further in view of Lasher et al. (U.S. Patent 4,863,247) (hereinafter "Lasher"), claims 6-12, 24-29, 31, 32, 48, 53, 54 and 55 as being unpatentable over Huppenthal, Hinds and Chen, and further in view of Stribaek et al. (U.S. Patent 7,181,484) (hereinafter "Stribaek"), claim 17 as being unpatentable over Huppenthal, Hinds and Chen, and further in view of Chen et al. (U.S. Patent 6,687,725) (hereinafter "Chen2"), and claims 46 and 47 as being unpatentable over Huppenthal, Hinds, Chen and Lasher and further in view of Stribaek. Applicants respectfully traverse these rejections for at least the following reasons.

Based on the remarks made in the Office Action mailed July 21, 2010, including in the Response to Arguments section of the Office Action, there appears to be confusion in regard to the term *instruction set* or the term *instruction set architecture*. For example, on page 3, the Examiner states, "The concept of the claimed invention is to combine a set of arithmetic operations (accumulate and multiplication arithmetic operations) into a single arithmetic instruction or invocation." Applicants first note that this statement ignores the plain language of Applicants' independent claims (e.g., independent claim 1), which (as previously presented) does not recite "combining operations into a single arithmetic instruction or invocation," but recites that multiple, specific operations are performed <u>in response to executing a single arithmetic instruction of a processor instruction set</u>. As indicated above, Applicants' claims have been amended to further clarify that this single arithmetic instruction is an instruction of a processor's instruction set architecture that is implemented in the processor itself (e.g., using circuitry and/or other means). The operations performed in response to executing this single arithmetic instruction include operations that are dependent on the previous execution of another single arithmetic instruction of the processor instruction set (e.g., adding implicitly a partial result from a previously executed single arithmetic instruction of the processor instruction set), and operations to store and use at least a portion of the generated result in a subsequent computation. Thus, the claims define a <u>relationship</u> between the execution of <u>two successive single arithmetic instructions</u> of the processor instruction set, and a <u>relationship</u> between <u>a currently executing single instruction</u> of the processor instruction set and <u>a subsequent computation</u>.

On page 4 of the Office Action mailed on July 21, 2010, the Examiner states, "Huppenthal and Hinds disclose the concept of chaining of arithmetic operations. This chaining concept is part of the architecture of the computing system disclosed within the Huppenthal prior art claimed invention. The chain operation is a legitimate operation within the instruction set of the Huppenthal prior art." **This statement has no basis in the prior art references themselves.** There is no description of a "chain operation" in Huppenthal's instruction set, nor of any <u>single arithmetic instruction</u> in Huppenthal's instruction set that invokes the operations cited by the Examiner. In fact, as discussed in

previous Responses and again below, the cited portions of Huppenthal describe methods for implementing functionality that is explicitly not supported by the instructions included in the processor's instruction set. Instead, in Huppenthal, each memory algorithmic processor includes one or more FPGAs that are configurable to perform functions that are not performed by one of the instructions of the fixed instruction microprocessor-based processors in the system, and these functions (once programmed in the FPGAs) are invoked using a write instruction of the processor's instruction set, not a single arithmetic instruction of the processor's instruction set architecture that is implemented in the processor itself.

In these and other remarks, the Examiner seems to imply that the claimed invention merely involves treating multiple operations as if they were performed by a single instruction or software macro. However, the plain language of the claims (as currently amended) requires that these operations be performed in response to executing a single arithmetic instruction of a processor instruction set architecture (i.e. an instruction set architecture implemented in the processor itself). It is clear from the plain language of the claims, the description in Applicants' specification, and even the title of Applicants' disclosure that the claimed invention is directed to implementation of a new processor instruction in a processor's instruction set architecture whose execution results in the performance of the operations and relationships recited in the claims. The Examiner's characterization of the claimed invention and his interpretation of the term *instruction set* (and/or *instruction set architecture*) are clearly inconsistent with the ordinary meaning of these terms, as they would be understood by those of ordinary skill in the art. For example, the reference book IEEE 100: The Authoritative Dictionary of IEEE Standard Terms (7^{th} ed. IEEE Standards Information Network/IEEE Press. 2000) includes the following relevant definitions:

> instruction set: The complete set of instructions recognized by a given computer or provided by a given programming language.
>
> instruction set architecture (2) An ISA defines instructions, registers, instruction and data memory, the effect of executed instructions on the registers and memory, and an algorithm for controlling instruction execution.

computer instruction: A machine instruction for a specific computer.

computer instruction set: The collection of computer instructions possible on a given computer.

machine instruction (1): An instruction that a machine can recognize and execute. (2) An instruction in the machine language of a particular processing unit of a computer.

processor architecture: The system-visible interfaces to a central processor, including its instruction set, stack and register structures, and trap and interrupt-handling methods.

Similarly, the textbook, <u>Computer Architecture: A Quantitative Approach</u> (Second Edition. David A. Patterson and John L. Hennessy. Morgan Kaufmann Publishers. 1996) includes the following, on page 4, "the term *instruction set architecture* refers to the actual programmer-visible instruction set." Similarly, page 69 refers to the instruction set architecture as "the portion of the machine visible to the programmer or compiler writer."

The Free On-line Dictionary of Computing (see, "Instruction Set Architecture." The Free On-line Dictionary of Computing. Denis Howe. 19 Oct. 2010. <Dictionary.com http://dictionary.reference.com/browse/Instruction Set Architecture>) defines the term *instruction set architecture* (ISA) this way: "(ISA) The parts of a processor's design that need to be understood in order to write assembly language, such as the machine language instructions and registers. Parts of the architecture that are left to the implementation, such as number of superscalar functional units, cache size and cycle speed, are not part of the ISA. The definition of SPARC, for example, carefully distinguishes between an implementation and a specification."

Similarly, Wikipedia includes the following description of the terms *instruction set* and/or *instruction set architecture*, "An instruction set, or instruction set architecture (ISA), is the part of the computer architecture related to programming, including the native data types, instructions, registers, addressing modes, memory architecture,

interrupt and exception handling, and external I/O. An ISA includes a specification of the set of opcodes (machine language), and the native commands implemented by a particular processor. Instruction set architecture is distinguished from the microarchitecture, which is the set of processor design techniques used to implement the instruction set. Computers with different microarchitectures can share a common instruction set. For example, the Intel Pentium and the AMD Athlon implement nearly identical versions of the x86 instruction set, but have radically different internal designs." (From Wikipedia, the free encyclopedia, the cited entry last modified on 12 October 2010 at 21:27.)

As illustrated in these example references, the terms *instruction set* and *instruction set architecture* have well-understood meanings in the art, and refer to the instructions that are native to (and implemented in) a particular processor, i.e. those for which opcodes (machine language instructions) are defined for and implemented in the processor itself. As noted above, Applicants' claims have been amended to clarify that the single arithmetic instruction of Applicants' claims is an instruction of the processor's instruction set architecture that is implemented (e.g., by circuits and/or other means) in the processor itself.

Applicants assert that the Examiner has not identified a single arithmetic instruction of a processor instruction set architecture (i.e. implemented in a processor) in any of the cited references whose execution causes the performance of the operations described in Applicants' claims, or results in the relationships between successive instruction executions recited in the claims. For example, Huppenthal describes a special purpose MAP processor that can be configured to perform functions that are not part of a processor's instruction set in response to executing a write instruction of the processor's instruction set. Hinds describes operations of a processor's microarchitecture that are used to implement a single arithmetic instruction (a floating point MAC instruction). Hinds describes feedback between operations of the microarchitecture that implement this single arithmetic instruction. Hinds does not describe implicit feedback (or an implicit relationship) between this single arithmetic instruction and a previously executed

single arithmetic instruction of the processor's instruction set architecture, as required by Applicants' claim, i.e. implicitly adding a partial result of a previously executed single arithmetic instruction without specifying this partial result as an operand of the currently executing single arithmetic instruction. Applicants again assert that neither of these references, nor their combination, teaches or suggests this new single arithmetic processor instruction of Applicants' claimed invention.

As discussed in detail below, the combination of references does not teach or suggest the single arithmetic processor instruction of Applicants' claims, the recited operations performed in response to execution of such an instruction, or the recited relationships between successive instruction executions, whether considered alone or in combination.

In the Response to Arguments section of the present Office Action, the Examiner submits, "The add-chaining operation and multiply-chaining operation discloses a combination (chaining) of accumulate and multiplication operations into a single arithmetic instruction. The Specification in paragraph [1076] states that the instruction performs multiply-accumulate-chaining operation, which combines (appears to be equivalent to prior art chaining) add-chaining and multiply-chaining in one operation in order to avoid the multiplier latency from the processing of operand(s) information between arithmetic operations. This statement appears to suggest the combination or chaining of two or more arithmetic operations with an implicit input or transfer of operand(s) between arithmetic instructions. Contrary to Applicant's assertions, the referenced prior art appears to disclose an equivalent operation as the claimed invention. The Examiner is not mischaracterizing the teachings of Applicant's Specification. The concept of combining arithmetic operations is disclosed in the specification and the referenced prior art. The concept of the implicit transfer of operands between instructions is disclosed in the specification and the referenced prior art. The concept of utilizing arithmetic operations in the generation of an encryption key for cryptographic calculations are disclosed in the specification and the referenced prior art."

Applicants assert that the Examiner's generalization of concepts allegedly taught by the prior art teach nothing about the specific limitations of Applicants' claims. Applicants' claims are not directed to "the concept of combining arithmetic operations", "the concept of the implicit transfer of operands between instructions" or "the concept of utilizing arithmetic operations in the generation of an encryption key for cryptographic calculations", as suggested by the Examiner. Instead, they recite specific limitations on specific arithmetic instructions of a processor's instruction set architecture that are simply not taught or suggested by the cited references. For example, Applicants' claims describe a relationship between a currently executing single arithmetic instruction of the processor instruction set architecture and a previously executed single arithmetic instruction of the processor instruction set architecture, not merely a collection of isolated arithmetic operations that are performed in response to a single initiated instruction. In addition, neither of the references teaches the implicit transfer of operands between two instructions of the instruction set architecture. At most, they teach feedback between operations within the implementation of a single instruction (e.g., for a function implemented using one or more FPGAs, or within the microarchitecture implementation of a single FMAC instruction).

By contrast, paragraph [1076] of Applicants' Specification states, in its entirety, the following:

> The umulxck instruction is an efficient way to support public-key computations. Back-to-back scheduling of multi-word multiplications and accumulations is often difficult using today's instruction sets due to the multiplier latency. The umulxck instruction performs the multiply-accumulate-chaining operation which combines add-chaining and multiply-chaining in one operation and avoids the multiplier latency. Using the umulxck instruction, and referring again to FIG. 5, the calculation of a 64x1024 bit partial product (y0*X) and the accumulation with a previous partial product (s16:s0), can be accomplished in the following 20 instructions:

```
set register k=y0;
umulxck 0,0;                // clear extended-carry register exc first
r0 = umulxck x0, s0;
 . . .
 . . .
. . .
```

```
r15 = umulxck x15, s15;
r16 = umulxck 0, s16;        // catch 64 carryout bits
r17 = umulxck 0, 0;          // catch last carryout bit
```

As described in Applicants' specification, the umulxck instruction is an instruction of the processor's instruction set architecture, and each single umulxck instruction in this example performs both multiply and accumulate operations and also implicitly allows for accumulating an additional partial result of a previous single umulxck instruction, i.e. without requiring additional add operations or specifying an additional operand for the additional accumulate operation. Specifically, a umulxck instruction of the form shown above (rd = umulxck rs1, rs2 - where rs1 and rs2 are source operands and rd is a destination operand), performs (rs1*k + rs2 + a partial result of a previous single arithmetic instruction, such as a previous umulxck instruction) without explicitly specifying the additional operand (the partial result of the previous single arithmetic instruction). The result register specified by operand rd receives the lower order bits of the result of this calculation, and the extended carry register (which is also not specified by any of the operands of the instruction) receives the upper order bits of the result of this calculation.

In the example referenced by the Examiner, the third single arithmetic instruction (r0 = umulxck x0, s0):

performs the calculations (x0*k + s0 + exc), where exc = 0 (because the second instruction above cleared exc)

stores the lower order bits of the result of this calculation in r0; and

stores the upper order bits of the result of this calculation in exc.

In this example, the fourth single arithmetic instruction (not shown, but implied to be (r1 = umulxck x1, s1):

performs the calculations (x1*k + s1 + exc), where exc now contains the upper order bits of the result of the calculation performed for the third instruction, as noted above;

stores the lower order bits of the result of this calculation in r0; and

stores the upper order bits of the result of this calculation in exc.

Similarly, each successive single umulxck instruction in this example performs a similar calculation on the explicitly specified source operands and also implicitly adds the contents of exc that were stored in exc by the previous umulxck instruction as the high order bits of the calculation made for that previous umulxck instruction.

As described in detail above, the Examiner's characterization of the teachings of Applicants' Specification is clearly incorrect. The cited portions of Applicants' Specification describe a specific instruction of the processor's instruction set (umulxck). When a single instance of this specific processor instruction is executed by the processor, the processor performs specific multiply and accumulate calculations, including the implicit addition of a partial result of a previously executed single instance of the umulxck instruction. In other words, as noted above, the execution of one of the processor instructions recited in Applicants' claims facilitates a feedback relationship between two successive single arithmetic instructions, not merely the performance of a collection of arithmetic operations to implement a single initiated instruction, or a general concept of operation chaining during execution of a single arithmetic instruction. Because execution of these instructions facilitates this feedback relationship, the chaining of successive single instances of this umulxck instruction can be used to perform (and accelerate) the multi-word multiplies and accumulates that are common in cryptography applications.

In the Response to Arguments section of the present Office Action, the Examiner submits, "Huppenthal discloses an architecture for chaining a number of arithmetic operations (such as multiplication, accumulation, and etc) to form a single arithmetic instruction. The single arithmetic instruction is initiated and the sequence of chained operations is performed with operand transfer controlled by the computing system architecture via the usage of a chain port mechanism. The chain port mechanism supplies operands to each successive arithmetic operation in the sequence. The chain port mechanism supplies operands without any support from the processor. Hinds discloses

the generation of a partial result (high-order or low-order bits). Huppenthal and Hinds disclose chaining a set of two or more arithmetic operations within a single arithmetic instruction and the generation of a partial result as the operand that is implicitly transferred between the arithmetic instructions." Applicants again assert that the Examiner is clearly mischaracterizing the teachings of both Huppenthal and Hinds in suggesting that the combination teaches Applicants' claimed invention.

Huppenthal is directed to a multiprocessor computer architecture incorporating a number of memory algorithmic processors ("MAP") in the memory subsystem or closely coupled to other processing elements (e.g., one or more <u>fixed instruction set</u> microprocessor-based processors) to enhance overall system processing speed. The memory algorithmic processor architecture is an assembly that contains field programmable gate arrays (FPGAs) functioning as the memory algorithmic processors. In other words, each memory algorithmic processor includes one or more FPGAs that are configurable to <u>perform various functions that are not performed by one of the instructions of the fixed instruction microprocessor-based processors in the system</u>. As described in the Summary section of Huppenthal, a MAP element can function autonomously from its host system (i.e. without processor intervention) <u>once its operands have been loaded</u>. MAP elements (which are not <u>processor instructions</u>, nor do they represent <u>circuitry to execute arithmetic instructions of a microprocessor instruction set</u>) can be chained together to forward operands from one to another when multiple MAP elements are working together to perform a very large function. **These operands are not transferred between <u>arithmetic instructions of a processor's instruction set</u>, as the Examiner suggests, but are transferred between MAP elements that are working together to perform a single, large function, i.e. a function that is <u>not defined in the processor's instruction set architecture</u>.** As described in detail in Huppenthal, the MAP architecture, and FPGAs thereof, perform functions in response to <u>commands written to them using a write instruction</u> that specifies a command and operands in the data. This is clearly not analogous to the limitations of Applicants' claims, which require execution of <u>single arithmetic instructions of a processor instruction set architecture</u> in succession. For example, Table 2 of Huppenthal (and the accompanying description)

describe commands that are communicated to the MAP architecture by issuing a write instruction from processor 12. These commands are used to configure various elements of the MAP architecture (e.g., RMB, RUC, RECON, LDROM), to pass operands to the MAP architecture (e.g., WRTOP, LASTOP), and to initiate the performance of a function implemented in the MAP circuitry (e.g., START).

While performing a function using the MAP circuitry of Huppenthal may involve performing a series of calculations or other functions using one or more FPGAs, this teaches absolutely nothing about Applicants' claimed invention. Applicants' claims are not directed to the general concept of performing a collection of arithmetic operations to carry out a single function, or even a single arithmetic instruction, as the Examiner implies, or to the chaining of operations performed within a MAP architecture or FPGA in response to a command issued using one or more <u>write instructions</u> of a processor, but to <u>specific arithmetic instructions in a processor's instruction set architecture</u>, individual instances of which <u>implicitly</u> add partial results of <u>a previously executed single arithmetic instruction in the instruction set</u> when executing a <u>current single arithmetic instruction</u> without explicitly specifying the partial result as an operand of the current single arithmetic instruction.

To establish a *prima facie* obviousness of a claimed invention, all claim limitations must be taught or suggested by the prior art. *In re Royka*, 490 F.2d 981, 180 U.S.P.Q. 580 (C.C.P.A. 1974), MPEP 2143.03. The cited art does not teach or suggest all limitations of the currently pending claims, some distinctive limitations of which are set forth in more detail below.

Regarding claim 1, the cited art fails to teach or suggest *in response to executing a single arithmetic instruction of a processor instruction set architecture implemented in a processor of the device, multiplying a first number by a second number; and adding implicitly a partial result from a previously executed single arithmetic instruction of the processor instruction set architecture to generate a result that represents the first number multiplied by the second number summed with the partial result, wherein the partial*

result comprises a high order portion of a result of the previously executed single arithmetic instruction, and wherein the single arithmetic instruction does not include an explicit source operand for specifying the partial result; storing at least a portion of the generated result; and using the stored at least a portion of the generated result in a subsequent computation in the cryptography application.

The Examiner submits, "Huppenthal discloses a method implemented in a device, the method comprising: executing a single arithmetic instruction of a processor instruction set, and storing at least a portion of the generated result; and using the stored at least a portion of the generated result in a subsequent computation in the cryptography application, <u>and wherein the single arithmetic instruction does not include an explicit source operand for specifying the partial result</u>" (emphasis Examiner's). Specifically, the Examiner cites Huppenthal (in column 3, lines 1-7) as disclosing "number of MAP elements chained together to accomplish a single function or operation (implies a single arithmetic instruction)." **The Examiner's remarks regarding the implication of a single arithmetic instruction of a processor instruction set are completely unsupported by the reference itself.** In fact, as discussed above, the memory algorithmic processors taught by Huppenthal are explicitly disclosed as being separate from the fixed instruction set microprocessors in the system (such as processors 12_1 – 12_n). Instead, they act as co-processors implemented in FPGAs or another type of user array to perform algorithmic functions in response to commands issued to them using a <u>write instruction</u> of the fixed instruction set processor, which is clearly not an <u>arithmetic instruction</u> of the fixed instruction set processor.

The Examiner cites Huppenthal (in column 3, lines 18-25) as disclosing that MAP elements can receive operands via chained port, and (in column 18, line 66 – column 19, line 3) as disclosing, "output data from one MAP element to be sent directly to the user array of the next MAP element with no processor intervention via a chain port." Applicants again assert that the chaining of MAP elements to perform a large function coded in FPGAs and initiated by a write instruction teaches nothing about the limitations of Applicants' claim. For example, it does not teach, in response to executing a <u>single</u>

arithmetic instruction of a processor instruction set architecture, multiplying a first number by a second number; and adding implicitly a partial result from a previously executed single arithmetic instruction of the processor instruction set. The chaining of MAP elements and passing of operands between MAP elements over a chain port when performing a single large operation implemented in one or more FPGAs, as described by Huppenthal, teach nothing about feedback between successive arithmetic instructions of a processor's instruction set architecture.

The Examiner states, "Huppenthal does not specifically disclose a multiplying and summing sequence" and relies on Hinds to teach such a sequence. Applicants again note that claim 1 does not merely require "a multiplying and summing sequence" but also requires feedback between two different arithmetic instructions of the processor instruction set, i.e. adding implicitly a partial result from a previously executed single arithmetic instruction when executing a current single arithmetic instruction. The Examiner submits that Hinds discloses multiplying a first number by a second number; and adding implicitly a partial result from an executed arithmetic instruction to generate a result that represents the first number multiplied by the second number summed with the partial result, wherein the partial result comprises a high order portion of a result of the previously executed single arithmetic instruction. Specifically, the Examiner cites Hinds (in column 3, lines 5-17) as disclosing, "applying a multiply-accumulate operation to operands; multiplying operands and adding multiplication results" and (in column 8, lines 6-25) as disclosing, "chained multiply-accumulate operation; carry save adders and usage of partial multiplier (partial results: high order bits); output of results of partial multiplier is normalized and passed to carry save adders and final product adder)." Hinds is directed to an apparatus and method for implementing a floating point MAC instruction that takes three operands as inputs. The cited portions of Hinds describe the operation of an individual floating point MAC instruction in the processor of Hinds, which is implemented (in the microarchitecture of Hinds' processor) using a multiplication and an addition operation. As described in Hinds, all three operands are explicitly specified for the MAC instruction; there is no additional operand that is implicitly added during execution of Hinds' MAC instruction, i.e. *adding implicitly a partial result from a*

previously executed single arithmetic instruction... wherein the single arithmetic instruction does not include an explicit source operand for specifying the partial result. The Examiner's citation in column 8 refers to a prior art "chained" multiply-accumulate FPU, illustrated in FIG. 3B, which is operable to implement a single floating point multiply-accumulate operation (A+(B*C)) or floating-point multiply-subtract operation (-A+(B*C)) in response to issuance of a <u>single floating-point instruction</u>. This "chained" multiply-accumulate FPU passes results of a partial multiplier to a carry save adder and final product adder <u>as part of executing a single floating point MAC instruction</u>. This partial result is not <u>implicitly added</u> as part of executing <u>a subsequent arithmetic instruction</u>, as in Applicants' claimed invention, nor is it <u>stored</u> and <u>used in a subsequent computation in the cryptography application</u>.

The Examiner submits that it would have been obvious to one of ordinary skill in the art to modify Huppenthal for a multiplying and summing sequence as taught by Hinds, stating, "One of ordinary skill in the art would have been motivated to employ the teachings of Hinds to specifically increase the speed of multiply-accumulate operations and achieve faster computations. (Hinds col. 1, lines 27-29)." The cited passage of Hinds describes "interest in developing FPUs arranged specifically to perform multiply-accumulate operations with increased speed." However, it is not clear how the Examiner means to incorporate the teachings of Hinds into the multiprocessor system of Huppenthal, or how that would improve performance of multiply-accumulate instructions, which are not described anywhere in Huppenthal. Does he mean to imply that the floating point MAC instruction of Hinds (and associated FPU logic) should be <u>added to the instruction set</u> (and processor circuitry) of the fixed-instruction processors (e.g., processor 12) in the system of Huppenthal? In this case, the Examiner's citation regarding the MAP architecture would not be applicable to this new instruction, since it would not be used to implement it. Or does the Examiner mean to imply that the FPU of Hinds could be implemented using the MAP architecture of Huppenthal (e.g., in one or more FPGAs) such that a floating point MAC operation is invoked by commands issued to the MAP architecture using a <u>write instruction</u>? Again, this would not teach or suggest

using the MAP architecture to implement an arithmetic instruction of the processor's instruction set, as the Examiner has previously suggested.

In addition, since the floating point MAC instruction of Hinds does not implicitly add a partial result from the previous execution of another floating point MAC (or a partial result from the previous execution of any other arithmetic instruction in the processor's instruction set), as suggested by the Examiner and as required by Applicant' claim, no combination of Huppenthal and Hinds would result in Applicants' claimed invention. At most, it could result in a system in which the floating point MAC operation of Hinds (which does not include such feedback) is implemented algorithmically in one or more FPGAs of the MAP architecture of Huppenthal, e.g., if such an instruction is not included in the instruction set of the fixed-instruction processors of Huppenthal.

The Examiner admits that Huppenthal-Hinds does not specifically disclose supporting a cryptography application and relies on Chen to disclose supporting a public-key cryptography application (citing Chen, column 6, lines 23-25, and "arithmetic operations to support acceleration of cryptographic functions"). The Examiner states, "It would have been obvious to one of ordinary skill in the art to modify Huppenthal-Hinds for supporting a cryptographic application as taught by Chen. One of ordinary skill in the art would have been motivated to employ the teachings of Chen to greatly improve the performance of cryptographic circuits. (Chen col. 5, lines 40-42)." The Examiner seems to imply that the reason to modify Huppenthal-Hinds to support cryptographic applications is to improve the performance of something that apparently does not exist in Huppenthal-Hinds (cryptographic circuits) for something that the Examiner admits that Huppenthal-Hinds does not support (i.e. cryptographic applications). Therefore, the Examiner has not provided a valid reason to combine the references. In addition, as described in detail above, the cited art does not teach all of the limitations of Applicants' claims, whether taken alone or in combination.

For at least the reasons stated above, Applicants assert that the Examiner has failed to establish a *prima facie* rejection of claim 1.

Independent claims 43, 57, and 64 include limitations similar to those recited in claim 1 and discussed above, and were rejected for similar reasons. Therefore, the arguments presented above apply with equal force to these claims, as well.

Independent claim 18 includes limitations similar to those recited in claim 1 and discussed above, and was rejected for reasons similar to those discussed above regarding claim 1. Therefore, Applicants traverse this rejection for at least the reasons presented above regarding limitations in this claim that are similar to those in claim 1.

In addition, claim 18 recites *adding a third number to generate a result that represents the first number multiplied by the second number summed with the partial result and the third number*. The Examiner submits that Hinds discloses these limitations using the same citations and reasoning as those included in remarks directed to claim 1. However, as discussed in detail above, Hinds teaches a floating point MAC instruction of the form (A+(B*C)), for which operands A, B, and C are explicitly specified. Hinds does not disclose a MAC instruction that <u>adds partial results of a previously executed instruction</u>, when executing this instruction, to *generate a result that represents the first number multiplied by the second number <u>summed with the partial result</u> and the third number*, as required by claim 18.

For at least the reasons above, the rejection of claim 18 is unsupported by the cited art and removal of the rejection thereof is respectfully requested.

Claims 50, 61, and 65 include limitations similar to those recited in claims 1 and 18 and discussed above, and were rejected for the same reasons as claims 1 and 18. Therefore, the arguments presented above apply with equal force to these claims, as well.

Applicants assert that numerous ones of the dependent claims recite further distinctions over the cited art. Applicants traverse the rejection of these claims for at least the reasons given above in regard to the claims from which they depend. However,

since the rejections have been shown to be unsupported for the independent claims, a discussion of the dependent claims is not necessary at this time. Applicants reserve the right to present additional arguments.

CONCLUSION

Applicants submit the application is in condition for allowance, and an early notice to that effect is requested.

If any fees are due, the Commissioner is authorized to charge said fees to Meyertons, Hood, Kivlin, Kowert, & Goetzel, P.C. Deposit Account No. 501505/6000-32301/RCK.

Respectfully submitted,

/Robert C. Kowert/
Robert C. Kowert, Reg. #39,255
Attorney for Applicant(s)

Meyertons, Hood, Kivlin, Kowert, & Goetzel, P.C.
P.O. Box 398
Austin, TX 78767-0398
Phone: (512) 853-8850

Date: October 21, 2010